IR Max User's Guide

Contents:

1. IR Code Basics

1.1. Introduction

To the casual user of a TV remote, Infrared communications almost seems magical. An invisible burst of Infrared light provides quick and reliable control of an appliance without needing wires. As with most modern technologies, its apparent simplicity hides many of the more complex techniques needed to make it happen. This section examines some of the techniques used to encode and reliably send IR commands.

1.2. IR Coding Basics

IR remotes communicate with their target devices by sending a burst of timed pulses of modulated infrared light using a specially designed infrared Light Emitting Diode (LED). A carrier frequency (often around 38 or 40 kHz) is turned on and off for precise durations and precise time intervals between the pulses. The use of a carrier frequency instead of just plain IR light enhances the noise and interference rejection characteristics of the system. It helps the receiver ignore other sources of IR light like sunlight and certain types of artificial lighting. The use of a carrier also improves the range (operating distance) of a remote by allowing the IR LED to conduct a higher current without overheating, since even during a long "ON" pulse, its actual duty cycle is only 50%.

Although the actual data coding format varies from one brand of remote controlled equipment to another, there are enough similarities between them that it becomes worthwhile to describe an actual format and then apply the knowledge gained to the others. One reason for this similarity is that many different manufacturers of remote controlled appliances often use the IR components of just a few chip set makers. One of these, and the one we will discuss, is the "NEC-1" 32 bit format. The NEC corporation of Japan was one of the first to market chip sets specifically for IR control and supplies many TV and audio component manufacturers. The NEC-1 format begins a code transmission with a long burst of IR followed by a long gap (called the *lead-in*), then with 32 bits of data, and then imposes a long gap (*lead-out*) before the transmission of any new code or repeat, allowing the receiver to determine that this is the beginning of a new transmission. The lead-out also supplies the "next pulse" needed to determine if the last data pulse is a binary 1 or a 0. The On and Off timing of the carrier is called the *signal envelope* and is what we're interested in here. Viewed on an oscilloscope, it would look like this (Fig. 1):

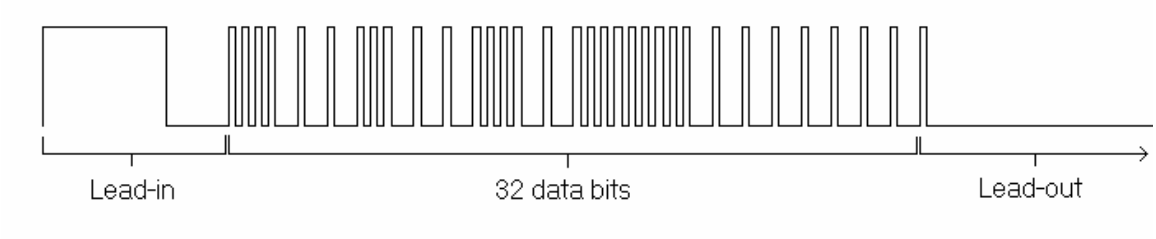

Fig. 1

You will notice that in the data bit area of Fig. 1, the pulses all have the same width but the time between any two pulses seems to be one of either two constant (long or short) values. It is this time difference between a pulse and the next one that determines if the first one is a binary 1 or a 0. Let's assume a short interval means a 0 and a long interval means a 1. Here is the code shown in Fig.1 with the data bit value written under each one (Fig. 2):

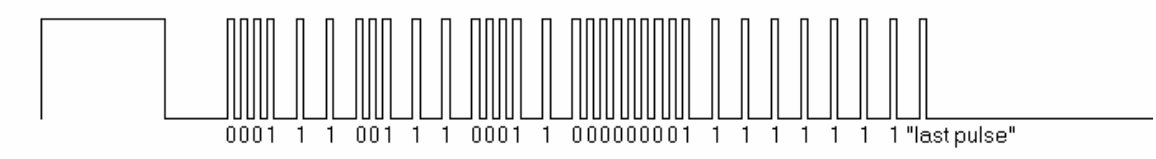

Fig. 2

Now let's write these 32 bits as 4 data bytes (Fig. 3):

```
0001 1100
1110 0011
0000 0000
1111 1111
```

Fig. 3

With a bit of observation, you will notice that the first two bytes have opposite bit patterns, as do the last two bytes. This is not a coincidence, but part of the error checking scheme for this protocol. Commonly, the NEC-1 protocol sends a device id (the first two bytes) followed by the command (last two bytes). What this particular example is saying is that it is sending code 00 for device 1C (hex). Other commands from the same remote will repeat the first two bytes because they are for the same device, but the last two bytes will change (and always complement each other) to specify the various commands that this remote can send.

The NEC-1 code is one of several that you might encounter when observing learned codes from various brands of remotes. When the NEC-1 code became too well used in the electronics industry, manufacturers of A/V equipment literally ran out of unique codes! The NEC Company responded by creating an expanded version (NEC-2) using 48 bits instead of 32, allowing for thousands of combinations by using 6 data bytes instead of just 4. To illustrate other variations you may come across, here is a sample of a Sony IR code (Fig. 4). This code uses a shorter lead-in pulse, has only 12 data bits, and varies the pulse width instead of the interval between pulses to distinguish between binary 0 and 1. Furthermore, the code is normally repeated 4 times for a button press (and the receiver expects the 4 repeats). For clarity, only two repeats are shown in Fig. 4

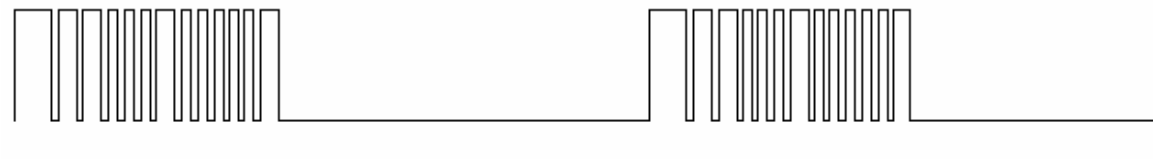

Fig. 4

This explanation of IR code formats is not meant to be an exhaustive tutorial on all the code formats that you might encounter, but is intended to give a general overview of how IR commands are constructed from timed bursts of modulated IR light. There are two important things to remember at this point: 1- IR transmitters and receivers use and expect a specific IR carrier frequency, and: 2- The signal envelope of the IR bursts provides a unique timing pattern that the receiver will use to identify each device and command.

1.3. Data Representation of IR Codes

When IR remotes were first available, they only controlled the appliance they came with. Then manufacturers started creating "universal" remotes that could control several devices of their own brand (e.g. Panasonic TV and Panasonic VCR). Then third party brand remotes became available that could control several different brands of equipment. As the availability of appliances multiplied and new models came out at an ever-increasing rate, these universal remotes quickly became obsolete. The next step was the "learning" remote that could memorize codes sent to them by the original appliance remote. This was a handy development but still required the original remote to be available so it was of little help when the learning remote was bought to replace a lost or broken original. The development that followed was a remote that could be programmed using a computer. This opened up other possibilities, such as creating or adding commands that the original remote didn't even have. One such remote that has become widely known is the Philips Pronto, and its format for representing IR codes as computer data has become a "common denominator" in areas where code exchanges are often happening, such as on internet user forums. The IR Max program also supports the importing of Pronto format codes. Because of this, we will take a brief look at the Pronto format, and then examine the ADI LIR format, in order to see how the two end up accomplishing the same task and ease the learning of the LIR format for someone who might already be familiar with the Pronto format.

1.4. Pronto Format

Here is the Pronto "learned IR" format representation of the same NEC-1 code that we used in the previous example (Fig. 5):

```
0000 006c 0000 0022 0157 00ad 0015 0016 0015 0016 0015 0016 0015 0041
0015 0041 0015 0041 0015 0016 0015 0016 0015 0041 0015 0041 0015 0041
0015 0015 0015 0015 0015 0016 0015 0041 0015 0041 0015 0015 0015 0016
0015 0016 0015 0016 0015 0016 0015 0016 0015 0015 0015 0015 0015 0041
0015 0041 0015 0041 0015 0041 0015 0041 0015 0041 0015 0041 0015 0041
0015 06be
```

Fig. 5

Remember the two essential things we needed to remember at the end of the "IR Code Basics"? Here they are again: 1- IR transmitters and receivers use and expect a specific IR carrier frequency, and: 2- The signal envelope of the IR bursts provides a unique timing pattern that the receiver will use to identify each device and command.

In the Pronto format, the carrier frequency is encoded in the 2nd field ("006c" in the example above, all fields are hexadecimal values). To calculate the carrier frequency, you simply need to do the following calculation: 4,145,146 / value of 2nd field. We must first convert 006c to decimal; which is 108. 4,145,146 / 108 = 38,380, or about 38.38 kHz. That takes care of the carrier frequency. The signal envelope data starts with the 4th field and goes all the way to the end. Each pulse is encoded as a pair of fields, with the first field giving the carrier "On" value and the second field the "Off" value. The actual numbers (always in hexadecimal) give the number of *carrier cycles* for each part of the pulse. The time duration of a carrier cycle is the reciprocal of the frequency: 1 / 38,380 = 26 uSec. Let's convert the first two pulses (field pairs) into actual time values (Fig 6)

```
0157 = 343(dec) * 26 uSec = 8918 uSec On
00ad = 173(dec) * 26 uSec = 4498 uSec Off
0015 = 21 (dec) * 26 uSec =  546 uSec On
0016 = 22 (dec) * 26 uSec =  572 uSec Off
```

Fig. 6

Now look at Fig. 1 again and you will notice that the relative time durations of these two pulses correspond quite accurately with the lead-in pulse and the first data pulse. If you continue and convert the entire code and then graph it, you will end up with a graph identical to Fig. 1.

1.5. LIR Format

As can be expected, the LIR format accomplishes our 2 basic requirements described in Code Basics, but using a different encoding method. A LIR code uses 256 bytes of memory, of which the first 200 are used to store the code data itself. Fig. 7 shows the same NEC-1 code previously discussed in LIR format:

```
77 fe e3 6f 90 0d 90 0d 90 0d 90 29 90 29 90 29 90 0d 90 0d 90 29 90 29
90 29 90 0c 90 0c 90 0d 90 29 90 29 90 0c 90 0d 90 0d 90 0d 90 0d 90 0d
90 0c 90 0c 90 29 90 29 90 29 90 29 90 29 90 29 90 29 90 29 90 7e 7e 7e
7e 7e 7e 7e 7e 70 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Fig. 7

The first byte (77 hex) encodes the carrier frequency. To calculate the actual frequency, we use the following formula: Freq (in kHz) = 4608 / (first byte + 1). If we first convert 77 hex to 119 decimal and then perform the calculation: 4608/ (119 + 1) we get 38.4. So the carrier frequency is 38.4 kHz.

The next 199 bytes encode the signal envelope timing. Each of these bytes actually contains two fields: The highest bit indicates whether the carrier should be On or Off, and the lower 7 bits indicate for how much time, in multiples of 40 uSec. Let's examine the first 5 timing bytes (Fig 8):

```
fe = carrier On for 7e time.  7e = 126(dec) * 40 uSec = 5040 uSec On
e3 = carrier On for 63 time.  63 =  99(dec) * 40 uSec = 3960 uSec On
6f = carrier Off for 6f time. 6f = 111(dec) * 40 uSec = 4440 uSec Off
90 = carrier On for 10 time.  10 =  16(dec) * 40 uSec =  640 uSec On
0d = carrier Off for 0d time. 0d =  13(dec) * 40 uSec =  520 uSec Off
```

Fig. 8

One thing you may have noticed right away: the first two data values both turn the carrier On. Since 7 bits can only hold a maximum time value of 126 (actually 127 but due to design considerations, data byte values of "7f" and "ff" cannot be used) any time interval lasting more then 126 multiples of 40 uSec simply use two or more data bytes to achieve the required total length. In this case, data values **fe** and **e3** add up to create our long lead-in pulse. If we add the two time values for the lead-in pulse, we get 9000 uSec. You can compare these pulse time values with the ones for the Pronto format (Fig 6) and you will see that they are within a few percent of each other. This is normal. Both the Pronto and the LIR formats have granularity (i.e.: minimum time interval multiples) and compensate for some of the computing overhead required like interrupt processing to generate the codes. This is not a problem however, since IR communications are not highly precise by nature and most receivers are designed to compensate for timing variations of up to 10% or more.

The complete definition of the LIR format is:

Byte 0 – carrier frequency data
Bytes 1 to 199 – signal envelope data
Bytes 200 to 223 – (unassigned)
Bytes 224 to 255 – IR code name

The carrier data value can be anywhere from 08 to FE, for carrier frequencies of 19 kHz to 512 kHz ? Note that a higher carrier value translates to a lower carrier frequency. Actual frequency is calculated by converting the data value to decimal and then using the formula:

Freq (in kHz) = 4608 / (first byte + 1)

Data values can range from 00 to 7E for carrier off time, and from 80 to FE for carrier on time, with the time value being represented by the lowest 7 bits and must be multiplied by 40 uSec to calculate the total actual time period. At the end of a code, you may notice a string of 7E data bytes followed a string of 00 values. The value of 7E turns off the carrier but still consumes the time interval to create a long lead-out time. Using 00 tells the transmitter not to spend any more time on this code and to move ahead to the next code if another one is queued to be sent.

2. Using IR Max

2.1. Introduction

IR Max is provided as a means of allowing users of the Adicon series of controllers and modules to customize and enhance the performance of the Infrared (IR) capabilities of these devices. Its purpose is to go beyond simply learning codes and to add the possibility of copying individual codes between Learned IR (LIR) files, copy and paste IR codes as plain text for easy exchange over a medium like the internet, and to allow actual editing of the codes themselves. A handy graphing function allows the user to visualize a code as if it were displayed on an oscilloscope, facilitating the correlation between a point in the pulse train and the actual data corresponding to it. Finally, a convenient conversion utility allows an IR code published in the Philips Pronto format (a widely used format for exchanging codes on the internet) to be imported and converted automatically to the LIR format.

When you first start up IR Max, you see one large screen as shown in Fig. 9:

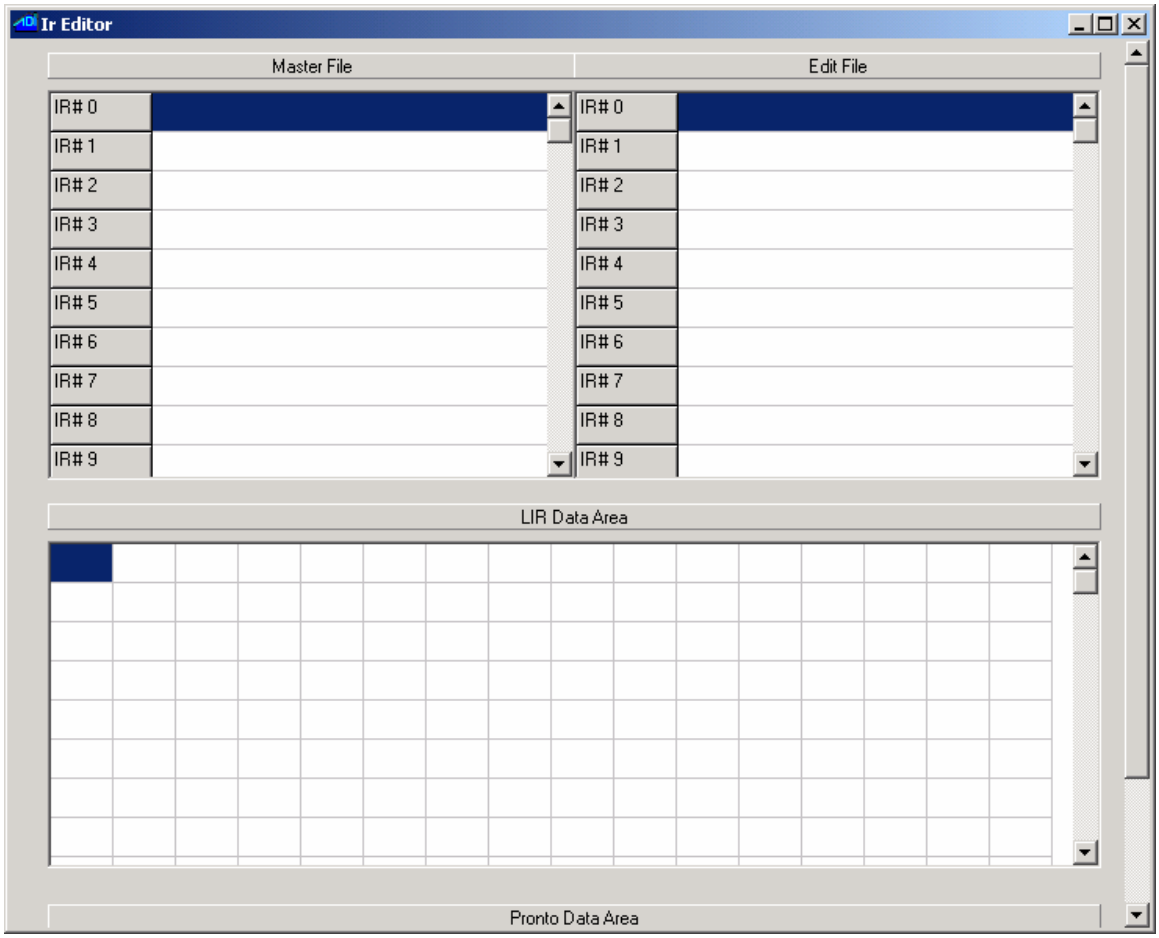

Fig. 9

There are 3 main areas on the screen. At the top is the file management area, divided in two columns. The rightmost column is labeled **Edit File** and is the one you use to edit, create and import codes to create or modify a LIR file. The **Master File** column on the left is used if you want to copy IR codes from an existing LIR file to the Edit File. Think of the **Master File** column as your source file for the codes while the **Edit File** is the destination.

The center section is the **LIR Data Area**. This part of the screen contains one edit box for each one of the 256 bytes that make up an IR code entry.

Finally, the bottom part of the screen is labeled as the **Pronto Data Area** (you may need to use the vertical scroll bar to view it completely). This is one large text field where you can paste a text representation of a Pronto code for converting and importing as a LIR code.

The use of each area will become self evident as you read the descriptions of the various operations that can be carried out with the utility. The best way to explain the usage of IR Max is by actual examples of typical editing operations, so lets get started.

2.2. Copying codes from one LIR file to another:

This is probably the most straightforward operation you can do and is a good way to gain familiarity with IR Max. Begin by choosing the existing LIR file that you want to take the codes from: Right click over the **Master File** area (top left of screen) and click on **Open** (…the only selection there is). A file-browsing menu will appear (Fig. 10)

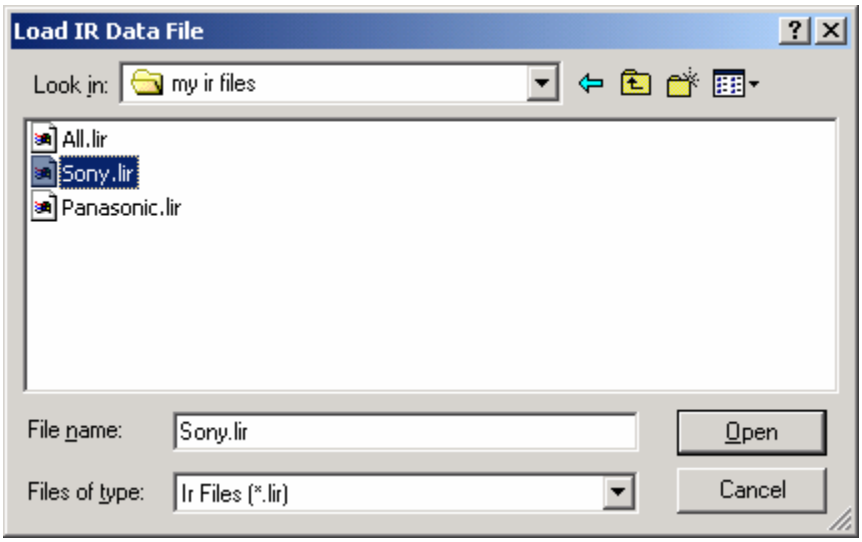

Fig. 10

Select the file that you want to import LIR codes *from*. Note that if you open a LIR file in which the individual codes have never been named, it is normal that you may see random characters appear in the name fields. This can be safely ignored. If you are creating a brand new LIR file, then you are ready to move on to the next step. If you are importing codes *into* an existing LIR file, then right click again, this time over the **Edit File** area (top right) and using the file-browser that will open, choose the LIR file that you want to import codes into.

To copy a single code, click once on the code in the **Master File** that you want to import. You will see the code highlighted. Now click on the code location under **Edit File** where you want to copy it to. You will see that location highlighted and the original code location under **Master File** will now display in reverse video. With the mouse cursor still over the destination location, right click and choose **Paste from Master File** and you will see the name (if any) appear in the new location. It is just as easy to copy a range of codes from the Master File: Click on the first code that you want to import under **Master File** and drag the mouse to the last code that you want to import. You can drag the mouse past the bottom or top edge of the Master File area to scroll through the code list. Once all the codes you want to import are highlighted, click on the first (lowest numbered) code location where you want to import the codes into the **Edit File** area and once again right click and choose **Paste from Master File**. You will see the range of codes now appear in the Edit File, starting with the code location you chose and up to the last location needed to accommodate the number of codes selected. Fig. 11 shows an example of selecting codes #3 through #8 from the Master File and pasting them as codes #1 an up in the Edit File:

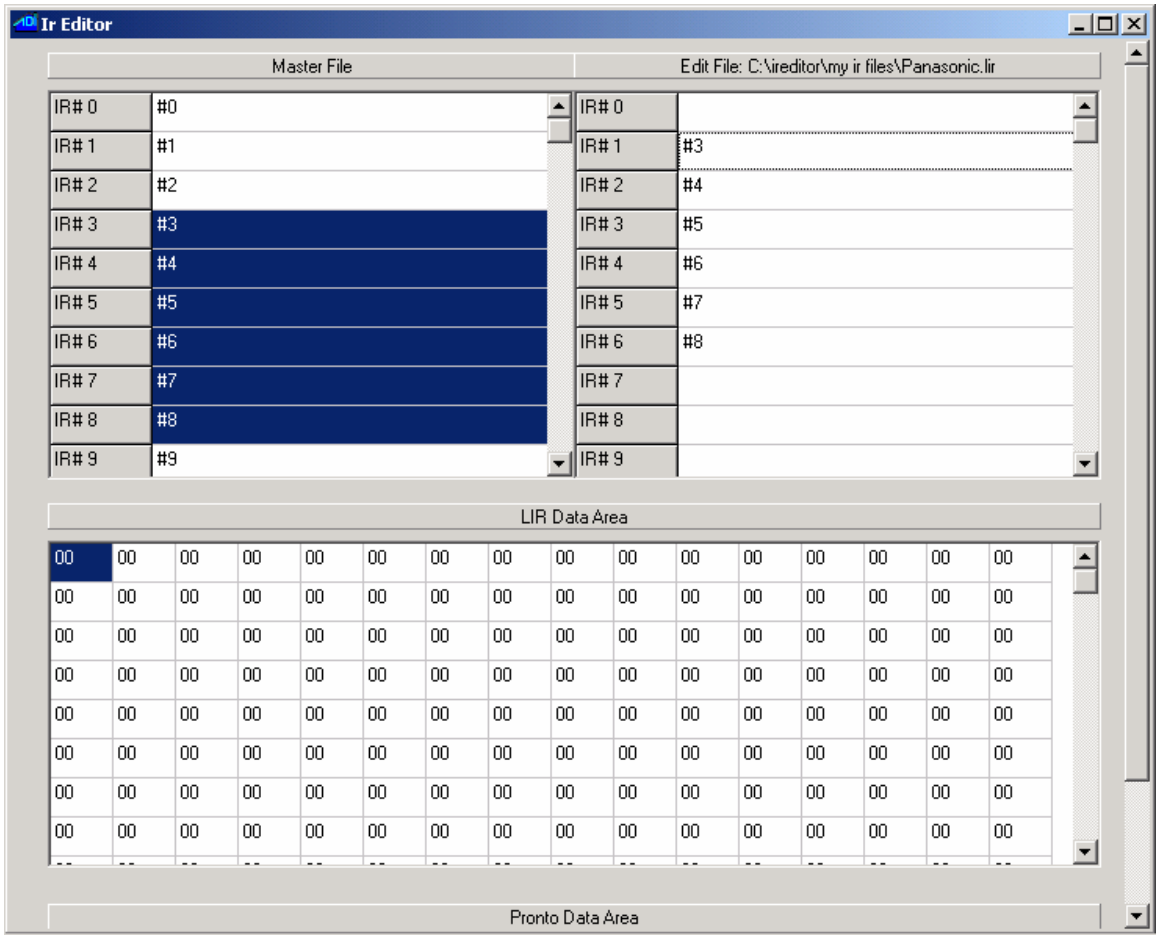

Fig 11

If your intent is to combine LIR codes from several different source LIR files and create one final destination LIR file, then you can copy the codes you want from a Master File as explained above, then open a different Master File and continue importing codes from that one and so on.

The code location names under **Edit File** can also be edited by clicking on each one and entering a name.

Once you have finished composing your Edit File, you right click over the **Edit File** area and choose **Save** or **Save As** to save it.

2.3. Editing LIR codes

You might have noticed (while copying and/or naming LIR codes) that when you clicked on a code location in the **Edit File** list, the data displayed in the **LIR Data Area** changed. This is in fact the way that you choose a code for manual editing: Open the LIR file that contains the codes that you want to modify under the **Edit File** list and then click on a code to display its hexadecimal data representation in the **LIR Data Area**. You can click on each data byte location individually and edit the raw data. The editing area will automatically prevent you from entering invalid data values like **7f** or **ff**. When you have finished editing a code and you want to apply your changes, you must import it back into the code location still highlighted in the **Edit File** by right clicking over its entry and choosing **Paste From LIR Data Area**. Be careful: do not accidentally *left click* the code location instead! To do so will cancel the changes that you made and reload the original data for that code location in the **LIR Data Area**! Of course, left clicking is also the way to cancel any changes you made to a code if you feel you have made an error and want to start over again.

Editing raw hexadecimal data is fine when you know exactly what you want to modify but it can be hard to visualize the final result or the exact part of the signal envelope you are editing just by looking at numbers. This is where the graphing utility can come in handy. With the cursor over the **LIR Data Area** or the **Edit File** area, right click and choose **Graph**. A separate window will pop up showing you an oscilloscope-like view of the signal envelope. Your screen will look like Fig. 12:

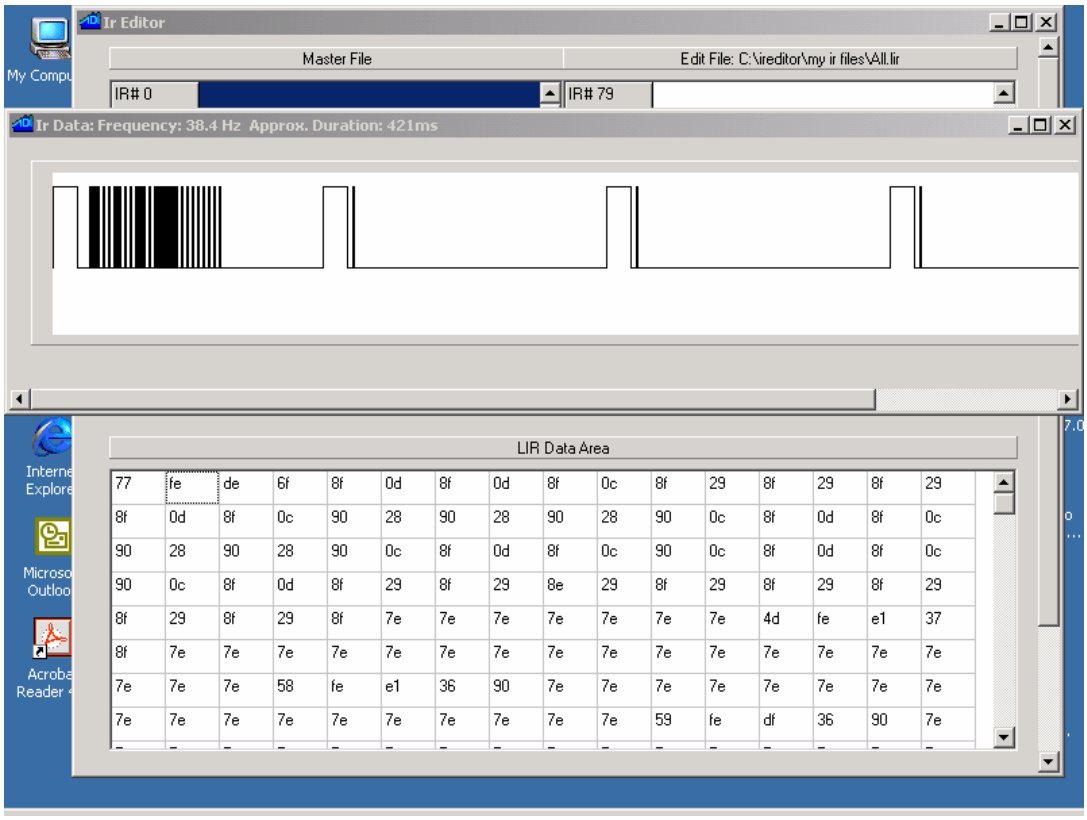

Fig. 12

The caption bar of the graph window shows you the carrier frequency calculated according to the first byte, and the total duration of the code in milliseconds. The solid black pulses that you see in graph in Fig 12 are caused by the individual pulses being so close together (over the displayed time scale) that they appear as being all stuck together. To see these better, you can use the zoom feature. Begin by clicking on the graph at the point where you would like to begin visual expansion. In our case, this is right at the beginning of the first, long lead-in pulse. When you click on a spot in the graph, the corresponding LIR data byte will be highlighted in the **LIR Data Area** below. Verify that the second data byte (**fe**) is highlighted. Now right click over the graph area and choose **Zoom IN** and the graph's time scale will be expanded by a factor of two (Fig. 13)

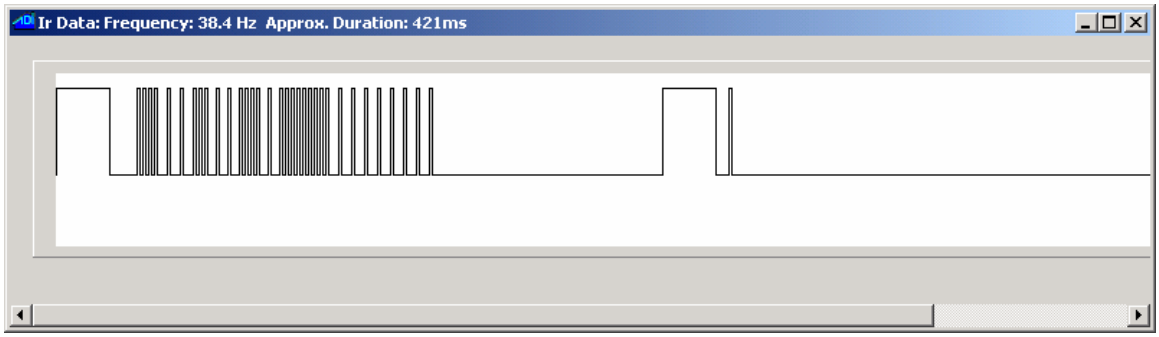

Fig. 13

If we again choose the beginning of the first pulse and **Zoom IN** one more time, we get Fig 14:

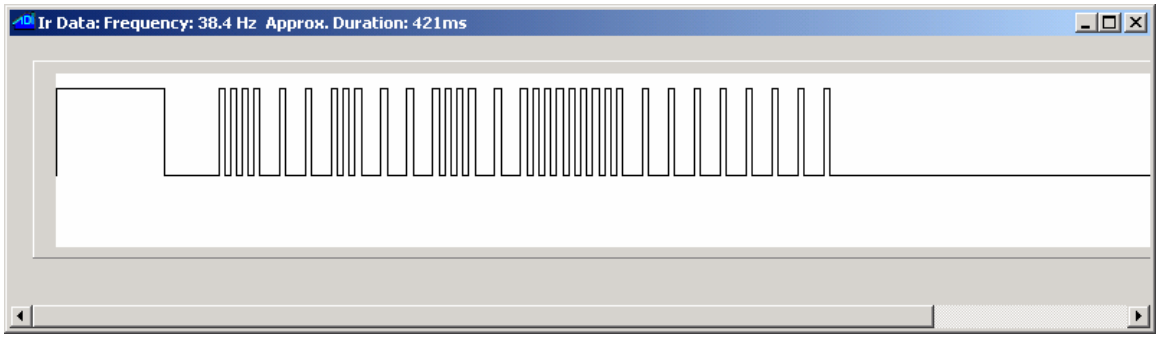

Fig. 14

Look familiar? This is the same IR code that is displayed in Fig. 1(in section 1). You will also notice that the right click menu allows you to zoom out or return to full size. As previously mentioned, clicking on a point in the graph display highlights that point's data location in the **LIR Data Area**. It also displays that point's location in time as an offset from the beginning of the code, useful for measuring pulse durations and overall code lengths. When a pulse or interval is long enough that two or more data bytes are needed to encode it, the correct data byte will always be highlighted, depending on which precise point of the long pulse or interval that you click on.

The ability to click on a point in the graph display and see which data byte it corresponds to is a handy feature when cleaning up learned codes. The term "cleaning up" is often used in reference to the process of removing repeats or partial codes that often occur when learning codes manually. Figures 12 and 13 show the repeats that are sent after the expiration of the lead-out time of the code itself (the repeat sequence is often a short code that tells the receiver to simply repeat the action specified by the original command). To remove these repeat commands, click on a point in the graph right at the end of the original code's lead out

time (i.e.: just before the lead-in pulse for the repeat code) and look at the corresponding byte in the LIR data area. You will usually notice that this will be at the end of a long string of **7e** data values. Now all you need to do is enter **00** for the data values from that point up to the end of the code data (the 200th byte in the code location).

The **LIR Data Area** only allows you to modify individual bytes; you cannot cut, copy or paste single or multiple bytes within that area. To do that, you can copy the entire data area to the Windows clipboard and then edit it as plain text in a text editing window or utility (such as "Notepad"), and then paste it back into the **LIR Data Area**. If you want to do this, right click over the **LIR Data Area** and choose **Copy as Text to Clipboard**. You can now paste the entire code to the text editor of your choice and perform your modifications there. Once the code is edited to your liking, copy the data bytes to the Windows clipboard and in the **LIR Data Area** right click menu, select **Paste from Clipboard Text**.

One trick that may be useful: you can use the **Pronto Data Area** at the bottom of the editor screen as a text editing area if you wish. Just paste the clipboard contents there and use it as a simple text editor, then copy the contents back to the clipboard and paste back to the **LIR Data Area**. The window might be a bit small for that usage but can still be useful for simpler edits. Note that if you accidentally click on **Convert** in the **Pronto Data Area** when using it this way, no harm will be done and you will get an **Invalid Pronto Format** message box.

If you edit LIR data externally by using the clipboard, remember that it is up to you to maintain a correct image of the code data. Be careful that the byte offset of the first byte for the name field does not get changed. *Remember*: no matter how you edit or import IR codes as LIR data, you need to use **Paste From LIR Data Area** in order to copy the code data into a code location in the **Edit File** area.

Copying and pasting LIR data using the clipboard has another useful purpose: this is how you can export and import codes in the LIR format for exchanging them as plain text over the internet in emails or on user forums such as the Adicon support forum. This is much easier then attaching entire LIR files just for the purpose of sending a few codes. A LIR code pasted as plain text appears in Fig. 7 of this manual. If you are reading this manual as a MS-Word document, you can actually copy the contents of Fig. 7 to your clipboard and import it as LIR data. Then as an exercise, graph it to recreate Figs. 12, 13 and 14.

2.3.1. Importing Pronto Format Codes.

IR Max allows you to import codes published in the Pronto format and convert them to LIR format. This function allows you to take advantage of discrete and special IR codes widely available from multiple sources, such as www.remotecentral.com (in the "Files" area). To use it, you copy the Pronto format code from your source (web page, forum, etc.) to your clipboard and then paste it to the **Pronto Data Area**. Fig. 5 shows a Pronto format code. You then click on the large **Convert** button in the **Pronto Data Area** and you will see the equivalent LIR format code in the **LIR Data Area**, where it can be edited and saved as explained in the previous paragraphs. Fig. 15 shows the Pronto code of Fig. 5 pasted in the editor and converted to LIR format:

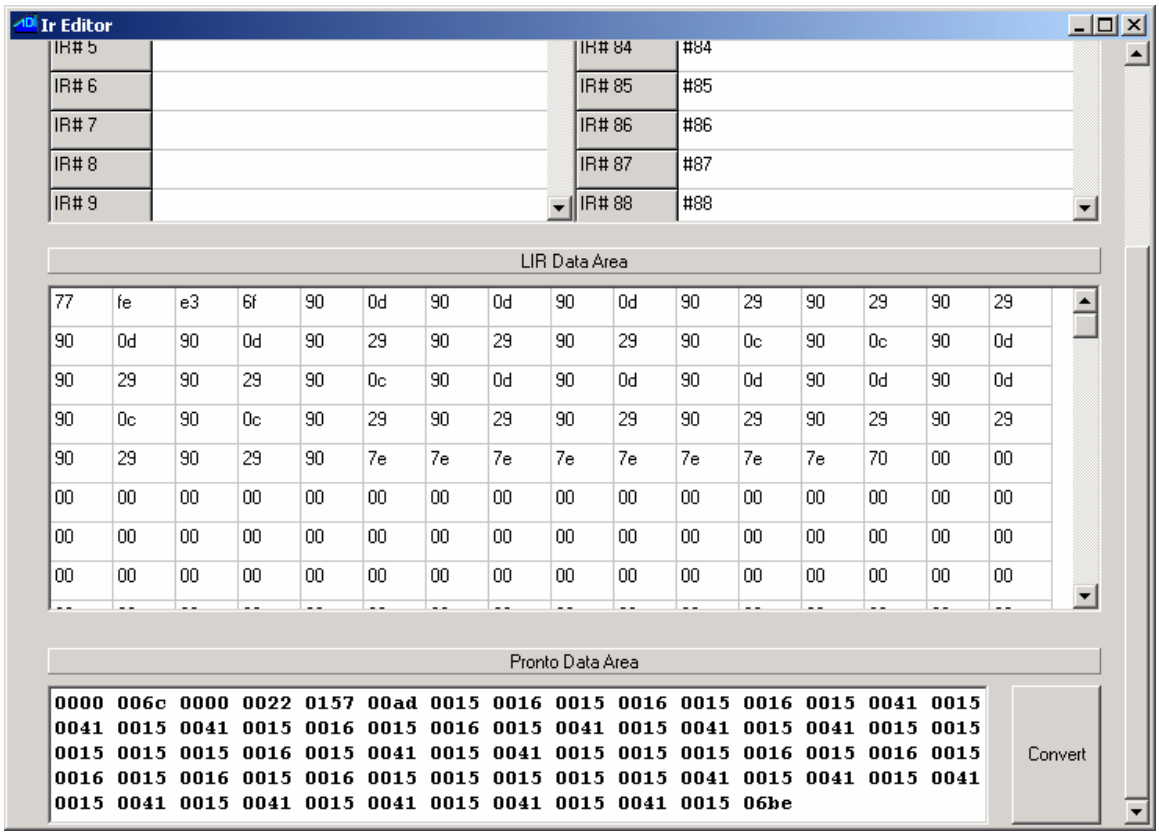

Fig. 15

As stated in the previous paragraphs, if you are reading this manual as a MS-Word document, you can actually copy the Pronto format code shown in Fig. 5 to your clipboard and import it as a Pronto code as an exercise. If the code you paste into the **Pronto Data Area** does not conform to the Pronto format and you try to convert it, you will get a message box displaying **Invalid Pronto Format**. Lastly, remember that you need to use **Paste From LIR Data Area** in order to copy the newly created LIR code data into a code location in the **Edit File** area, so you should have a code location already selected when you import the Pronto code. If a code selection was not selected, selecting one will overwrite the LIR data with the current contents of that location. If that happens and you had just imported a Pronto code, just click on **Convert** again to recreate it.

3. Conclusion

IR Max allows the user to have total control over the contents of a LIR file. You can optimize the performance of your ADI controller or SECU16IR module by cleaning up and creating or importing IR codes in either the LIR or Pronto formats. You can also exchange codes with other users in an easy to manage text format.